dplyr - eine Kurzeinführung

Ulrich Reubold

12. Juli 2019

Das Package `dplyr` gegebenenfalls bitte installieren (sollte schon passiert sein, sofern Sie die Übungen bearbeitet haben):

```
install.packages("dplyr")

library(dplyr)
zweit = read.table(file.path(pfadu, "zweit.df.txt"))
```

1. Grundfunktionen in `dyplr`: die sogenannten "Verben"

Ein Grundprinzip von `dyplr` ist es, komplexe Aufgaben in eine Abfolge einfacher Aufgaben aufzuteilen, und diese einfachen Aufgaben auf besonders effektive Art und Weise zu lösen. Ein möglicherweise zunächst als Nachteil empfundener Effekt hiervon ist, dass der `dplyr`-Code für eine Aufgabe in `R` zunächst als (unnötig?) lange Kette von Befehlen erscheint; der große Vorteil hiervon ist jedoch, dass eine komplexere Aufgabe "in Worte gefasst" wird, und man auf diese Weise vergleichsweise leichtverständliche Teilaufgaben, die quasi-natürlichsprachlich formuliert sind, einfach kombinieren kann. Ein wichtiger Teil dieser Prozesses ist der Umstand, dass die eigentlichen Aufgaben als "Verben" verstehbar sind. Die für uns wichtigsten dieser "Verben" sind:

`dplyr`-Verb	Beschreibung
`select()`	wähle Spalte(n)
`slice()`	wähle Reihe(n)
`filter()`	filtere Reihe(n)
`arrange()`	ordne Reihen um
`mutate()`	erzeuge neue Spalten
`summarise()` bzw. `summarize()`	fasse mehrere Werte zu einem zusammen
`group_by()`	gruppiere nach Faktorstufen

Die für unseren Kurz wichtigen Verben sind `group_by()`, gefolgt von entweder `mutate()` oder `summarise()`.

Die genannten Befehle können nur mit Dataframes umgehen, und sie geben als Ergebnis immer einen Dataframe aus (der manchmal `tibble` genannt wird und etwas anders aussieht, da u.a. die Datentypen der Spalteninhalte angezeigt werden; das muss uns an dieser Stelle aber nicht weiter kümmern: ein `tibble` ist nur eine moderne Variante eines Dataframes).

Diese “Verben” können als Teil eines “Satzes” verwendet werden (innerhalb einer sogenannten Pipe, s.u.), aber auch als eigenständige Befehle; im letzteren Fall ist das erste Argument der Dataframe, mit dem etwas gemacht werden soll.

1.1 Beispiele

1.1.1 select()

```
head(zweit)
```

```
##   Vpn G  Ses  gpa l1score l2exposure l2score
## 1  S1 F  mid 9.58     124         70     210
## 2  S2 M  mid 4.94     105         80     172
## 3  S3 M  mid 6.18      96        100     180
## 4  S4 F high 6.09     103        130     186
## 5  S5 M  low 9.31      98          0     177
## 6  S6 F  mid 6.23      90         90     174
```

```
zweitVpnG = select(zweit,Vpn,G) # wähle aus dem Dataframe "zweit" die Spalten "Vpn" und "G"
head(zweitVpnG)
```

```
##   Vpn G
## 1  S1 F
## 2  S2 M
## 3  S3 M
## 4  S4 F
## 5  S5 M
## 6  S6 F
```

1.1.2 slice()

```
slice(zweit,1:3) #die ersten drei Zeilen
```

```
##   Vpn G Ses  gpa l1score l2exposure l2score
## 1  S1 F mid 9.58     124         70     210
## 2  S2 M mid 4.94     105         80     172
## 3  S3 M mid 6.18      96        100     180
```

1.1.3 filter()

```
filter(zweit,l1score>=120) # nur die Zeilen, in denen ein l1score größer oder gleich 120 vorkommt
```

```
##   Vpn G Ses  gpa l1score l2exposure l2score
## 1  S1 F mid 9.58     124         70     210
## 2  S7 F mid 6.86     121        110     191
## 3 S20 M mid 6.16     121         70     179
## 4 S72 M mid 7.24     122        120     193
## 5 S73 M mid 7.70     120          0     179
```

1.1.4 arrange() (und Gebrauch von desc())

```
zl1up = arrange(zweit,l1score) # ordne nach den Werten in l1score, aufsteigend
head(zl1up)
```

```
##    Vpn G  Ses  gpa l1score l2exposure l2score
## 1  S68 F high 7.51      69         60     173
## 2  S53 M high 6.01      78        120     174
## 3  S11 M  mid 8.47      81         90     178
## 4 S100 M  mid 3.63      81         30     157
## 5  S28 F  low 7.21      82        100     169
## 6  S50 M  mid 5.91      83         50     159
```

Durch `desc()` (für "descending" im Sinne von "descending order") kann man die Reihenfolge umkehren:

```
zl1down = arrange(zweit,desc(l1score)) # ordne nach den Werten in l1score, absteigend
head(zl1down)
```

```
##   Vpn G Ses  gpa l1score l2exposure l2score
## 1  S1 F mid 9.58     124         70     210
## 2 S72 M mid 7.24     122        120     193
## 3  S7 F mid 6.86     121        110     191
## 4 S20 M mid 6.16     121         70     179
## 5 S73 M mid 7.70     120          0     179
## 6 S24 F mid 4.28     118         70     183
```

1.1.5 mutate()

```
zweitdiff = mutate(zweit,diffl2l1=l2score-l1score) # erzeuge eine neue Spalte namens "diffl2l1"", die die Differenz der Werte in den Spalten "l2score" und "l1score" enthält
head(zweitdiff)
```

```
##   Vpn G  Ses  gpa l1score l2exposure l2score diffl2l1
## 1  S1 F  mid 9.58     124         70     210       86
## 2  S2 M  mid 4.94     105         80     172       67
## 3  S3 M  mid 6.18      96        100     180       84
## 4  S4 F high 6.09     103        130     186       83
## 5  S5 M  low 9.31      98          0     177       79
## 6  S6 F  mid 6.23      90         90     174       84
```

1.1.6 summarise() (bzw. summarize())

Das "Verb" `summarise()` wendet eine Funktion an, um aus vielen Werten einen zu erzeugen. Dies funktioniert aber nur mit den Funktionen `first()`, `last()`, `nth()`, `n()`, `n_distinct()`, `IQR()`, `min()`, `max()`,`mean()`,`median()`,`var()`,`sd()`; die wichtigsten davon (für diesen Kurs) sind:

```
summarise(zweit,min(l1score)) # zeige den kleinsten Wert der Spalte l1score
```

```
##   min(l1score)
## 1           69
```

```
summarise(zweit,max(l1score)) # zeige den größten Wert der Spalte l1score
```

```
##   max(l1score)
## 1          124
```

```
summarise(zweit,mean(l1score)) # zeige das arithmetische Mittel der Werte der Spalte l1score
```

```
##   mean(l1score)
## 1      100.9667
```

```
summarise(zweit,mean(l1score)) # zeige den Medianwert der Spalte l1score
```

```
##   mean(l1score)
## 1      100.9667
```

```
summarise(zweit,sd(l1score)) # zeige die Stichprobenstandardabweichung der Werte der Spalte l1score
```

```
##   sd(l1score)
## 1    10.59168
```

```
summarise(zweit,IQR(l1score)) # zeige den Interquartilsabstand (also den Abstand zwischen dem 75%-Quantil und dem 25%-Quantil) der Werte der Spalte l1score
```

```
##   IQR(l1score)
## 1        14.25
```

1.1.7 group_by()

Die Gruppierung, die durch `group_by()` vorgenommen wird, sieht man so nicht, denn es wird einfach der Dataframe in `tibble`-Form wieder ausgegeben, ohne das irgendetwas verändert wurde:

```
group_by(zweit,Ses) # gruppiere nach den Faktorstufen in Ses: low, mid, high
```

```
## # A tibble: 120 x 7
## # Groups:   Ses [3]
##    Vpn   G     Ses     gpa l1score l2exposure l2score
##  * <fct> <fct> <fct> <dbl>   <int>      <int>   <int>
##  1 S1    F     mid    9.58     124         70     210
##  2 S2    M     mid    4.94     105         80     172
##  3 S3    M     mid    6.18      96        100     180
##  4 S4    F     high   6.09     103        130     186
##  5 S5    M     low    9.31      98          0     177
##  6 S6    F     mid    6.23      90         90     174
##  7 S7    F     mid    6.86     121        110     191
##  8 S8    F     mid    4.33      86         70     168
##  9 S9    M     mid    5.71     106        100     185
## 10 S10   M     low    4.72      97         60     164
## # … with 110 more rows
```

`group_by()` wird aber dann höchst sinnvoll, wenn man etwas pro Gruppe machen will:

```
summarise(group_by(zweit,Ses),mean(l2exposure)) #berechne pro Faktorenstufe in Ses das arithmetische Mittel der Werte in der Spalte l2exposure
```

```
## # A tibble: 3 x 2
##   Ses   `mean(l2exposure)`
##   <fct>              <dbl>
## 1 high                92.4
## 2 low                 56
## 3 mid                 64.8
```

Da der Ausdruck `summarise(group_by(zweit,Ses),mean(l2exposure))` sehr lang ist, und daher etwas schwer verständlich, ist es einfacher, so etwas in Unteraufgaben aufzuteilen. Ein solches Verfahren - das sehr ähnlich wäre wie die Aufteilung in Unteraufgaben, wie wir sie aus dem package `ggplot2` kennen - gibt es auch in dplyr.

2. Modularisierung mittels des "Pipe"-Operators `%>%`

Man kann eine Aufgabe wie "berechne pro Faktorenstufe in Ses das arithmetische Mittel der Werte in der Spalte l2exposure" in *einen* komplexen Ausdruck wie `summarise(group_by(zweit,Ses),mean(l2exposure))` schreiben, kann dessen Teile aber auch einzeln aufführen, und diese durch die Verwendung des "pipe"-Operators `%>%` zu einem Ganzen zusammenführen:

```
summarise(group_by(zweit,Ses),mean(l2exposure))
```

```
## # A tibble: 3 x 2
##   Ses   `mean(l2exposure)`
##   <fct>              <dbl>
## 1 high                92.4
## 2 low                 56
## 3 mid                 64.8
```

```
#oder besser lesbar:

zweit %>% # nimm den Dataframe "zweit"
  group_by(Ses) %>% # teile diesen in Gruppen (basierend auf den Faktorstufen
in Spalte "Ses") ein und tue alles Nachfolgende pro Gruppe
  summarise(mean(l2exposure)) # berechne das arithmetische Mittel der Werte
in der Spalte "l2exposure"
```

```
## # A tibble: 3 x 2
##   Ses   `mean(l2exposure)`
##   <fct>              <dbl>
## 1 high                92.4
## 2 low                 56
## 3 mid                 64.8
```

Die generelle Schreibweise ist also

`dataframe $>$ funktion(Spalte)`

mit der Bedeutung: "Nimm aus dem Dataframe `dataframe` die Spalte `Spalte` und führe damit die Funktion `funktion` aus"; ohne Modularisierung sähe der Befehl folgendermaßen aus:

`funktion(dataframe,Spalte)`

Man kann - wie die Unteraufgaben in ggplot2 auch - beliebig viele einzelne Aufgaben kombinieren; wegen der Lesbarkeit kann man diese einzelnen, durch `$>$` verknüpften Befehle untereinander schreiben, man kann aber auch alles in eine Zeile packen:

```
zweit %>%
  group_by(Ses) %>%
  summarise(mean(l2exposure))

## # A tibble: 3 x 2
##   Ses   `mean(l2exposure)`
##   <fct>              <dbl>
## 1 high                92.4
## 2 low                 56
## 3 mid                 64.8

# oder

zweit %>% group_by(Ses) %>% summarise(mean(l2exposure))

## # A tibble: 3 x 2
##   Ses   `mean(l2exposure)`
##   <fct>              <dbl>
## 1 high                92.4
## 2 low                 56
## 3 mid                 64.8
```

Eine etwas komplexere Aufgabe könnte z.B. so aussehen:

“Berechnen Sie die gruppenspezifischen (Spalte Ses) Stichprobenstandardabweichungen der Differenzen aus den Scores für die Zweit- und die Erstsprache (Spalten l2score bzw. l1score) im Dataframe ‘zweit’!”

Diese Aufgabe kann man folgendermaßen in einfache Unteraufgaben aufteilen:

a) nimm den Dataframe ‘zweit’
b) errechne eine neue Spalte mit dem Namen ‘diffl2l1’, die die Differenzen der Spaltenwerte in ‘l2score’ und ‘l1score’ enthält
c) gruppiere nach den Faktorstufen in Ses
d) errechne die Stichprobenstandardabweichung für diffl2l1

In modularisierender `dplyr`-Schreibweise:

```
zweit %>% # a)
  mutate(diffl2l1 = l2score - l1score) %>% # b)
  group_by(Ses) %>% # c)
  summarise(sd(diffl2l1)) # d)

## # A tibble: 3 x 2
##   Ses   `sd(diffl2l1)`
##   <fct>          <dbl>
```

```
## 1 high            9.31
## 2 low             9.05
## 3 mid             9.20
```

N.B.: in diesem Fall wären b) und c) auch vertauschbar:

```
zweit %>% # a)
  group_by(Ses) %>% # c)
  mutate(diffl2l1 = l2score - l1score) %>% # b)
  summarise(sd(diffl2l1)) # d)

## # A tibble: 3 x 2
##   Ses    `sd(diffl2l1)`
##   <fct>           <dbl>
## 1 high             9.31
## 2 low              9.05
## 3 mid              9.20
```

3. Anwendungsfälle in diesem Kurs

Beispiel 1: Differenzbildung für einen gepaarten t-Test (mittels `group_by()` und `summarise()` mit der Funktion `diff()`)

Wenn man einen gepaarten t-Test durchführen muss, sollte man vor der Abbildungserstellung die Differenz zwischen den zwei Werten pro Sprecher errechnen. Z.B.:

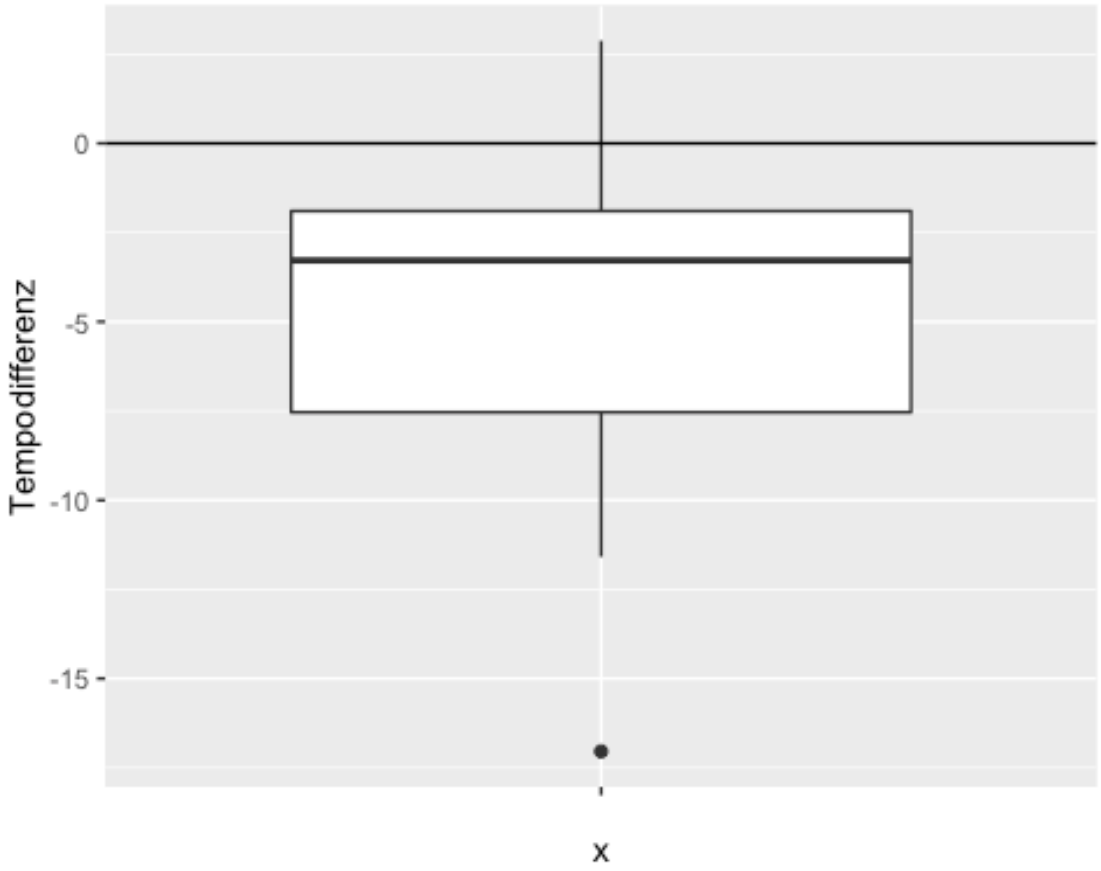
Die Daten in `fremd` zeigen fuer 30 Sprecher (Vpn) eine Messung der Sprechgeschwindigkeit (tempo), wenn sie in ihrer Muttersprache oder ihrer Zweitsprache (Sprache) reden. Pruefen Sie durch eine Abbildung und statistischen Test, inwiefern die Sprechgeschwindigkeit von der Sprache beeinflusst wird!

```
pfadu = "http://www.phonetik.uni-muenchen.de/~jmh/lehre/Rdf"

fremd = read.table(file.path(pfadu, "fremd.df.txt"))

#########################################################
fremd.diff = fremd %>%
  group_by(Vpn) %>%
  summarise(tempo = diff(tempo))
#########################################################

library(ggplot2)
ggplot(fremd.diff) +
  aes(y = tempo, x = "") +
  geom_boxplot() +
  geom_hline(yintercept = 0) +
  ylab("Tempodifferenz")
```




```
shapiro.test(fremd.diff$tempo)

##
##  Shapiro-Wilk normality test
##
## data:  fremd.diff$tempo
## W = 0.95534, p-value = 0.2344

# fremddiff$tempo ist normalverteilt, und daher duerfen wir einen t-Test
anwenden:
t.test(fremd.diff$tempo)

##
##  One Sample t-test
##
## data:  fremd.diff$tempo
## t = -5.3138, df = 29, p-value = 1.06e-05
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -6.264775 -2.782559
## sample estimates:
## mean of x
## -4.523667
```

Beachten Sie: dieses Beispiel hätten Sie auch ohne `dplyr` mittels des Befehls `aggregate()` lösen können:

```
fremd2.diff = aggregate(tempo ~ Vpn, data = fremd, FUN = diff)

head(fremd2.diff)

##   Vpn  tempo
## 1  S1  -8.95
## 2 S10   2.58
## 3 S11 -10.51
## 4 S12  -4.47
## 5 S13 -17.04
## 6 S14  -2.85

#zum Vergleich das Ergebnis der dplyr-Lösung:
head(fremd.diff)

## # A tibble: 6 x 2
##   Vpn    tempo
##   <fct>  <dbl>
## 1 S1     -8.95
## 2 S10     2.58
## 3 S11   -10.5
## 4 S12    -4.47
## 5 S13   -17.0
## 6 S14    -2.85
```

Beispiel 2. Proportionen errechnen (durch zählen von `TRUE`s mit `sum()` mittels `summarise()`, sowie Erstellung einer neuen Spalte `p` (die dann die Proportionen enthält) mittels `mutate()`)

In diesem Experiment haben Hörer aufgrund von F1-Änderungen entschieden, ob sie /a/ oder /a:/ wahrgenommen haben. Wird die Wahl zwischen /a, a:/ von F1 beeinflusst?

```
f1 = read.table(file.path(pfadu, "adaten.df.txt"))

f1$P = f1$V == levels(f1$V)[2]
f1$Q = !f1$P

#########################################################
f1.sum = f1 %>%
  group_by(F1) %>%
  summarise(P = sum(P),Q = sum(Q)) %>%
  mutate(p = P/(P+Q))
#########################################################

head(f1.sum)

## # A tibble: 6 x 4
##      F1     P     Q     p
```

```
##   <int> <int> <int> <dbl>
## 1   220     0     1 0
## 2   300     1     2 0.333
## 3   340     1     3 0.25
## 4   360     0     4 0
## 5   380     0     8 0
## 6   400     2     9 0.182
```

4. Link zum `dplyr`-cheat sheet

https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf